

REMARKS

Claims 1, 12, 20, 31, 39, 40, 41, 42 and 46 have been amended, and claims 47 and 48 have been added. No new matter has been added. Claims 1-48 are pending.

In the Office Action, claims 12, 31, 39 and 42 are objected to for using the phrase "an second". These claims have been amended to correct this typographical error.

Independent claims 1, 20, 40 and 41 have been amended for clarification. These claims recite that the first meta parameter values in the encapsulated function are mapped to the first function call parameter values useable in the first computing environment. It is believed that no new issues requiring further search or consideration have been introduced by this amendment, as the Office Action reflects an understanding of the original claims that is equally applicable to the claims as amended. Additionally, the phrase "that cannot natively access the second application programming interface" has been repeated in the body of claim 1 to clarify the nature of the first process. This phrase was recited in the preamble of claim 1 as originally submitted.

In the Office Action, claims 1-3, 5-10, 20-22, 24-29, 40, 41, 43, 45 and 46 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Combs et al., US 6,766,348 ("Combs"). This rejection is respectfully traversed.

Claim 1 recites a method for accessing a second application programming interface in a second computing environment from a first process operating in a first computing environment, wherein the first process that cannot natively access the second application programming interface. In the example given in the application and recited in several dependent claims, the first computing environment is a Java® environment and the second computing environment is a C environment.

In the method of claim 1, a first function call made by a first process in the first computing environment is detected, and based on the first function call, an encapsulated function call is generated for transfer from the first computing

environment to the second computing environment. The encapsulated function call contains first meta parameter values mapped to first function call parameter values useable in the first computing environment. As described in the application, meta parameter values are intermediate or universal representations of the first function call parameter values and are exchangeable between the first and second computing environments. The encapsulated function call containing the first meta parameter values is transferred from the first computing environment to the second computing environment, where the meta parameters can be used to invoke a corresponding second function call.

Thus, the method of claim 1 is directed to a system, for example, in which a user wishes to execute a task from within the first computing environment that will require execution of one or more functions from within the second computing environment, but where the functions of the second computing environment conform to a second application programming interface (such as a C language API) that is not natively accessible to a process in the first environment with which the user interacts (such as a Java® server). By the use of the meta parameter values that are mapped to the parameter values of the first function call, the function in the second computing environment becomes accessible from the first computing environment.

Combs shows a method and system for allocating distributed resources of a computer network to application programs running on computers also attached to the network. A distributed resource allocator system comprises a number of identical processes running on the computers of the network. Application programs request allocation of resources from a local distributed resource allocator system process using a resource allocator applications programming interface ("RA-API"). Application programs request allocation of resources from a remote distributed resource allocator system process via the RA-API and a resource allocator access protocol ("RAAP").

In particular, there are three basic types of information exchange that occur in the context of the resource allocation system. Users call functions

provided by the RA-API that are executed by a resource allocator system agent ("RASA") to acquire, relinquish, and use computer resources. In the case that a user runs on a different computer than the RASA to which the function call is directed, the function call is packaged into a RAAP message for transport to the RASA that will execute the function call. Data returned to the user in response to the execution of the function call by the RASA is packaged into a return RAAP message and transported back to the user. Finally, each RASA maintains a global network information database ("GNID") that describes the state of the RAHS as a whole. When a RASA executes a function of the RA-API that causes the state of the RAHS to change, for example, a function that allocates a resource to a particular user, that RASA must inform the remaining RASAs of the RAHS that the resource has been allocated to the user, so that each of the remaining RASAs can appropriately update its GNID. The RASAs communicate among themselves using a resource allocator system protocol ("RASP").

Referring to Figure 5 of Combs, in the case of a remote resource allocation request, an RA user communicates with a Remote Access Agent User via the RA-API. The Remote Access Agent User in turn communicates with a Remote Access Agent System via the RAAP, and the Remote Access Agent System communicates with the resource allocator system using the same RA-API used between the RA user and the Remote Access Agent User. This functionality is described beginning at column 5, line 36 of Combs. It is emphasized that "when a user requests services from a RASA running on a remote computer, it makes those requests using the very same RA-API as it would use to make requests of a RASA running on the local computer system..." Moreover, it is apparent from Figure 5 and elsewhere in Combs that the resource allocator system employs the very same RA-API to communicate with the Remote Access Agent System as it uses to communicate with a local RA user.

The Office Action refers to Combs' teaching regarding a Bind function call and related arguments including user credentials, service priority, etc.. These elements are simply one piece of the overall RA-API. When an RA-user

executes a Bind function call, it is provided to the RA system as the very same Bind function call, whether the RA-user is local or remote.

It is respectfully submitted that Combs cannot render the method of claim 1 obvious under 35 U.S.C. § 103(a), because Combs does not teach or suggest all the elements thereof. In particular, Combs does not teach or suggest a method involving a process in a first computing environment that cannot natively access an application programming interface in a second computing environment, nor the use of meta parameter values that are mapped to first parameter values from the first computing environment.

With respect to the relationship between first and second computing environments, the Office Action is understood to be reading these on two computers in Combs, one having a remote RA user and the other having the Remote Access Agent System and the Resource Allocator System. However, the same applications programming interface, the RA-API, is used in both places. Thus, the remote RA user can natively access the API in the Resource Allocator System. There is no translation of the RA-API function calls generated by the RA user into another API. There is only one API shown in Combs, and it is the only API used by both the RA user and the RA system. The use of the RAAP to transport a function call from a remote user in no way changes the API used at either the RA-User end or the RA system end.

With respect to the meta parameter values, the Office Action states that these are not taught in Combs, but asserts that their use would be obvious "because it would have provided the capability for facilitating the transfer of function calls and data from application programs running on the local computer to the remote computer." It is respectfully noted that RAAP already provides for the transfer of function calls and data from the remote RA users to the RA system, and does so without resorting to the use of meta parameter values. Of course, no meta parameter values are necessary, because the same API is used at both ends. Thus, the encapsulation function of Combs merely has to place the elements of the RA-API function calls, unaltered, into an RAAP message for

sending to the RA system, and the de-capsulation function merely has to re-create the same RA-API function calls from the unaltered contents of the RAAP message. There is no need in Combs for meta parameter values.

Although the Office Action also states that Combs' Bind function and related arguments suggest meta parameter values, it is not seen how Combs provides any such suggestion. As explained above, the Bind function is simply one function of several that constitute the RA-API. There is no conversion of Bind function arguments either into or from meta parameter values that are mapped to the Bind function arguments. The Bind function executed by an RA User is native to the RA-API of the Resource Allocator system, and thus no meta parameter values are either present or needed in Combs.

Based on the foregoing, it is respectfully urged that claim 1 is not rendered obvious by Combs, and accordingly is allowable under 35 U.S.C. § 103 notwithstanding Combs' teaching.

Independent claims 20, 40 and 41, as well as all the dependent claims of claims 1, 20, 40 and 41, incorporate the above-discussed features of claim 1, and therefore are allowable in view of Combs for at least the same reasons discussed above.

Before the second rejection in the Office Action is addressed, it is noted that new claims 47 and 48 have been presented that are similar to claims 1 and 20 but include additional descriptive language that is believed to further clarify the distinctions from Combs. Claims 47 and 48 recite that the first function call includes a first function call parameter, and that neither the first function call nor the first function call parameter are usable as generated by the first process to access the second application programming interface. These claims further recite the use of an application programming interface definition defining (i) a mapping of the first function call and the first function call parameter to corresponding meta parameters, wherein a first meta parameter is an intermediate representation of the first function call and is mapped to a second function call defined in the second application programming interface, and a

second meta parameter is an intermediate representation of the first function call parameter and is mapped to a second function call parameter usable with the second function call to access the second application programming interface. Combs is not seen to teach or suggest such features. In particular, Combs is not seen to show a first function call parameter that is not usable to access a second API, and is represented by a meta parameter mapped to a second function call parameter that is usable to access the second API. As described above, the RA user and RA system of Combs both employ the same RA-API, and therefore any function call parameters generated by the RA user can be used to access the API of the RA system. As Combs lacks this particular functionality recited in claims 47 and 48, these claims are believed to be allowable in view of Combs.

In the Office Action, claims 4, 11, 12-19, 23, 30-39 42 and 44 are rejected under 35 U.S.C. § 103(a) as being unpatentable in view of Combs and Saulpaugh et al., US 6,298,345 ("Saulpaugh"). This rejection is respectfully traversed.

Independent claim 12 recites a method for automatically generating applications allowing operation of an application programming interface in a second computing environment from a first process in a first computing environment, in which the first process is not natively compatible with the second computing environment. A second application programming interface definition associated with a second computing environment is analyzed to discover second function definitions. Based on the analysis, string generators and parsers are automatically generated for each second function definition discovered in the second application programming interface definition. A first string generator encapsulates a first function call from a first process in the first computing environment into an encapsulated function call. A second parser receives and parses the encapsulated function call to invoke a corresponding second function definition in the second application programming interface definition for operation within the second computing environment. A second string generator encapsulates an output from the second function call from a second process in

the second computing environment into an encapsulated response, and a first parser receives and parses the encapsulated response to return the output to the first function in the first process operating in the first computing environment.

It will be appreciated that the method of claim 12 is responsible for automatically generating blocks of code, for example, that implement the functionality of the string generators and parsers. It can be embodied as a "program for generating programs". Such a method frees an application developer from the need to design, code and maintain string generator and parser programs. Rather, an application developer need only maintain items used by the method, such as a set of grammar definitions, API processor, and certain functions and data structures such as shown in Figure 9 and described in the corresponding text of the application. With these items, the method can be employed to automatically generate the string generators and parsers required to implement an API access method such as the method of claim 1 for arbitrary sets of APIs.

The teaching of Combs has been summarized above. As noted in the Office Action, Combs does not teach any automatic generation of string generators or parsers. Although the Office Action has referred to Figure 9 and column 18, lines 37 et seq. of Combs as showing string generators and parsers, this characterization is seen as misleading. The items in Figure 9 are simply data items including strings; no methods for generating the strings are shown. Furthermore, there is nothing in Combs that automatically generates the functionality recited in column 18, which includes the use of the RAAP protocol to carry RA-API function calls from a remote RA user to the RA system. All of this functionality is simply presented as part of the disclosure of Combs without any hint of being automatically generated.

Saulpaugh discloses a method and system for transforming an intermediate database form into an object-oriented database. The intermediate form is derived from a grammatical form of the object-oriented database through compilation. The grammatical form is an expression of an object-oriented

database in a textual form according to a grammar. The intermediate form comprises an array of intelligent entry objects that encapsulate data with methods for manipulating that data, but lacks the infrastructure of the object-oriented database. The intermediate form can be used to populate the object-oriented database with entries via a public API for accessing the object-oriented database. The object-oriented database is an object-oriented configuration database which stores configuration parameters pertaining to the software and hardware of a computer system, such as application programs, device drivers, system services, and other components. Saulpaugh's method is used to address the problem of non-persistence of the object-oriented database. The object-oriented database is rendered persistent by the process of "serializing" the object-oriented database into the grammatical form. It is then re-constituted by compiling the grammatical form into the intermediate form, via which the object-oriented database can be populated.

Col. 17, lines 19-26 of Saulpaugh, which are referred to in the Office Action, describes the compilation of the grammatical form database into the intermediate form as transforming information expressed in a first language to information expressed in a second language by applying one or more grammars to interpret and/or analyze the first form and create the second form. This description is understood to describe only a process for generating one form of database from another form. Saulpaugh does not describe or mention processes in different computing environments having different APIs, or the need for a method of enabling a process in one of the environments to access a non-native API in the other environment, and certainly does not describe any automatic generation of any such methods.

It is respectfully submitted that Combs and Saulpaugh cannot render the method of claim 12 obvious under 35 U.S.C. § 103(a), because neither of these references teaches or suggests the automatic generation of string generators or parsers that perform the various functions of encapsulating function calls from one computing environment and invoking corresponding function definitions in

the other computing environment as set forth in claim 12. The Office Action candidly states that Combs fails to teach any such automatic generating. And Saulpaugh fails to overcome this deficiency of Combs. Saulpaugh teaches only a process of compiling one database form into another, which is not what is recited in claim 12. Moreover, although the Office Action states that the use of Saulpaugh's teaching in Combs would facilitate the exchange of data between users of computation resources connected to a network and a resource allocation system that manages the use of those resources, there is no description of how Saulpaugh's teaching would achieve such facilitating. The various elements shown in Combs, including the RA-API and RAAP for example, already facilitate such data exchange. There is no need for the function of Saulpaugh in Combs. Furthermore, there is no discussion in Combs of any problem of non-persistence of a database, and certainly no description of a grammatical form from which an object-oriented database can be re-constituted. Thus, Saulpaugh's teaching is essentially irrelevant with respect to Combs as well as to claim 12.

Based on the foregoing, it is respectfully urged that claim 12 is not rendered obvious by Combs and Saulpaugh. Independent claims 31, 39 and 42, as well as all the dependent claims of claims 12, 31, 39 and 42, incorporate the above-discussed features of claim 12, and therefore are allowable in view of Combs and Saulpaugh for at least the same reasons discussed above.

In view of the amendments and remarks herein, it is respectfully urged that all the claims of this application are allowable in view of Combs, Saulpaugh and the other art of record. Favorable action is respectfully requested. The Examiner is urged to telephone the undersigned attorney to resolve any issues that may be remaining after this amendment.

No fee is believed to be required. If the U.S. Patent and Trademark Office deems a fee necessary, this fee may be charged to the account of the undersigned, Deposit Account No. 50-0901.

-33-

If the enclosed papers or fees are considered incomplete, the Patent Office is respectfully requested to contact the undersigned collect at (508) 366-9600, in Westborough, Massachusetts.

Respectfully submitted,

  
\_\_\_\_\_  
James F. Thompson  
Attorney for Applicant(s)  
Registration No.: 36,699  
CHAPIN & HUANG, L.L.C.  
Westborough Office Park  
1700 West Park Drive  
Westborough, Massachusetts 01581  
Telephone: (508) 366-9600  
Facsimile: (508) 616-9805  
Customer No.: 022468

Attorney Docket No.: EMC01-09(01044)

Dated: November 30, 2004